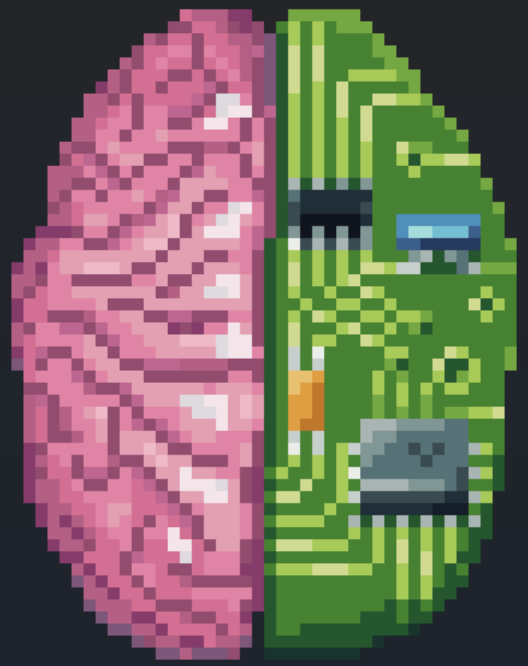
Brains in Games
–
Artificial intelligence in videogames



Jiří Hanák


IN-GAME
ARMA III


LANDS


ARMA
REFORGER

Foreword – meaning of ‘AI’

- ‘AI’ - a bit distorted term
  - Same word, different meanings – depends on who you ask
- In this presentation, AI:
  - doesn’t mean a LLM, a tool for content generation
  - means a controller for non-player entities in video games

Not really talking about ChatGPT

- Not this:

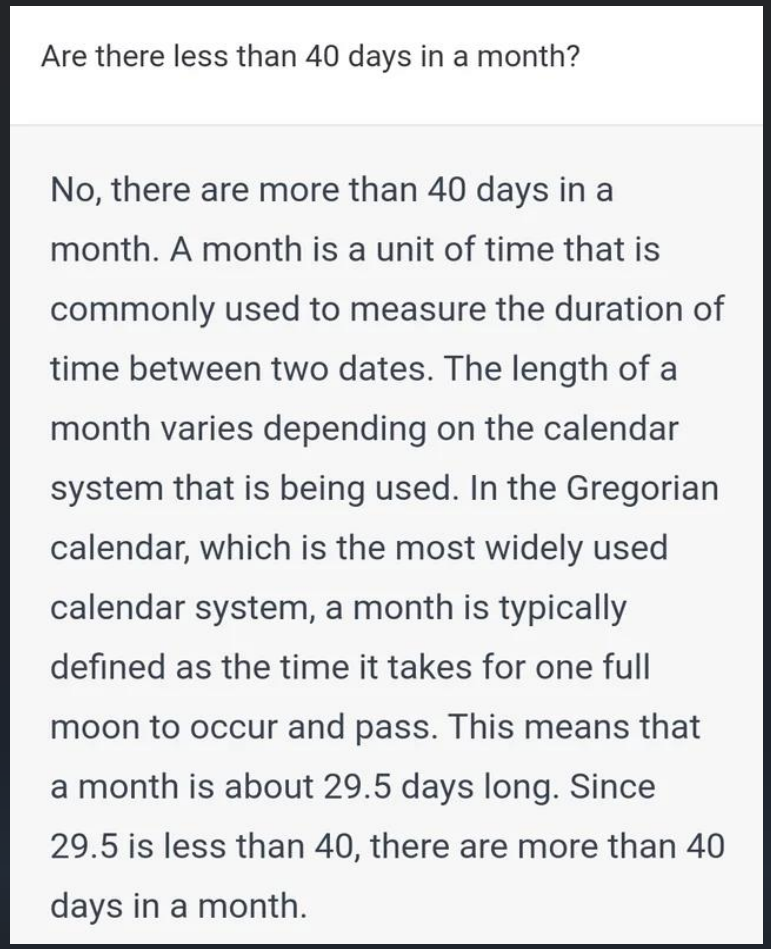


Fig 1. LLM spewing nonsense. ChatGPT, OpenAI, 2024

Not really talking about ChatGPT

- But this:

Fig 2. A situation that can actually happen in a game with robust AI. FarCry 5, Ubisoft, 2018

What is ‘game AI’?

- System for controlling in-game entities

What is ‘game AI’? - Entities

- System for controlling in-game entities

Fig 3. ‘AI entities’ can mean a horde of zombies. Days Gone, Bend Studio, 2021

What is ‘game AI’? - Entities

- System for controlling in-game entities

Fig 4. AI is often used for controlling ‘baddies’, like the group of cultists here. Far Cry 5, Ubisoft, 2018

What is ‘game AI’? - Entities

- System for controlling in-game entities

Fig 5. But AI can control friendly entities just as well - like this bunch of cute slimes. Slime Rancher 2, Monomi Park, 2022

What is ‘game AI’?

- System for controlling in-game entities
  - Movement / pathfinding
  - Combat
  - Interaction with world / other entities
  - Cooperation / Competition
- Goal
  - Good ‘actors’
  - Breathing life into the world
  - Fun experience

Fig 6. Controlling enemy entities – like this group of gnolls in Baldur’s Gate - is what game AI is very often about. Baldur’s Gate 3, Larian, 2023

Specific demands of games

- Performance considerations
  - Running in real time – AI needs to react to player & world changes
  - 60 FPS is ~16ms of CPU per frame – total budget , machine dependent :(
- Design considerations
  - Tuning behaviours is necessary – the goal is a fun game and that often involves challenge
  - Deterministic vs reactive
    - other aspect of fun can be overcoming rulesets
  - Every game's different

Fig 7. Overcoming AI rules can be very satisfying for tough enemies that are otherwise hard to beat – if you play it right, Commander O'Neill can follow you into deadly geysers of scarlet rot. Elden Ring, FromSoftware, 2022

Specific demands of games

- Performance considerations
  - Running in real time – AI needs to react to player & world changes
  - 60 FPS is ~16ms of CPU per frame – total budget , machine dependent :(
- Design considerations
  - Tuning behaviours is necessary – the goal is a fun game and that often involves challenge
  - Deterministic vs reactive
    - other aspect of fun can be overcoming rulesets
  - Every game's different
- Player expectations
  - AI needs to behave in a believable manner
  - AI needs to provide a challenge that's 'just right'

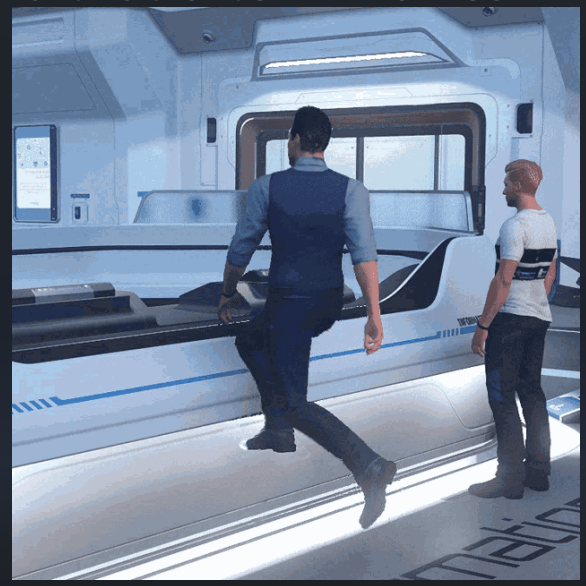
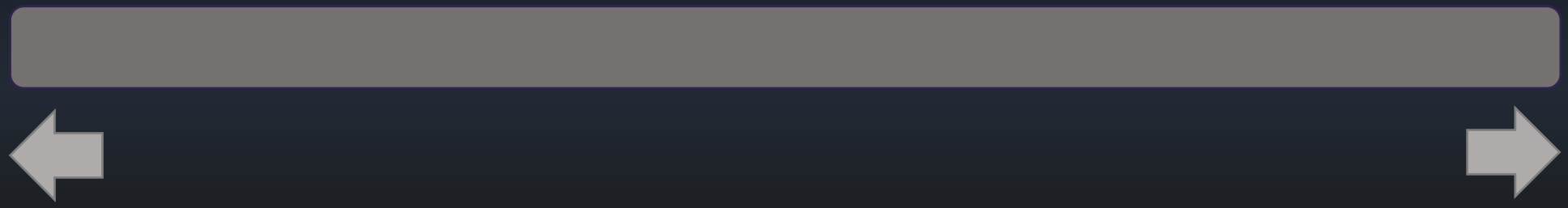


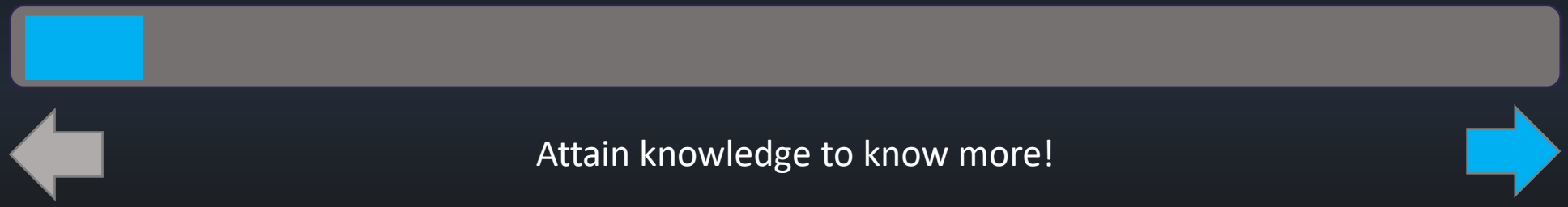
Fig 8. An NPC walking into an obstacle. Starfield, Bethesda, 2023

Before we start…



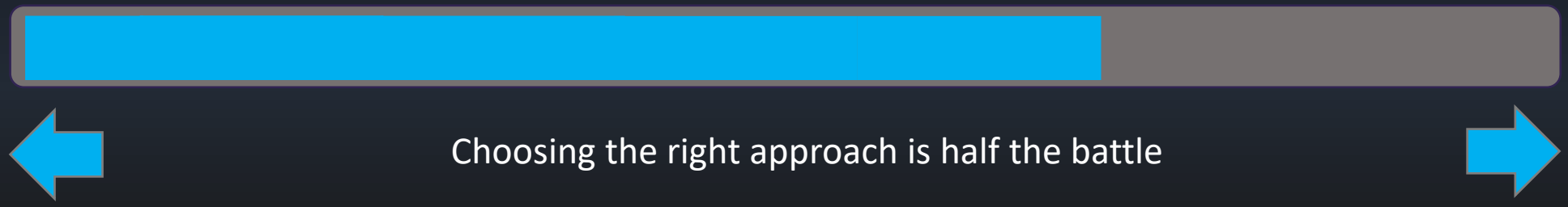
Before we start…

- Multiple techniques, we'll scratch the surface



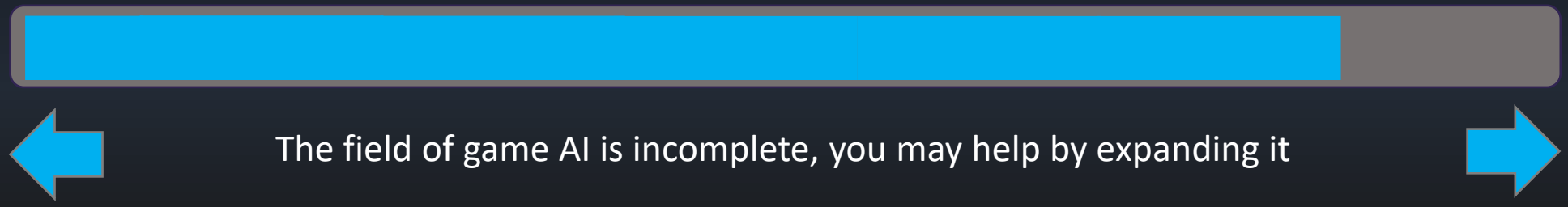
Before we start…

- Multiple techniques, we'll scratch the surface
- No generally correct approach – complex implementation != fun (or successful) game
  - When you're holding a hammer, everything looks like a nail
    - Learn to realize when you need to use a screwdriver
      - (or when duct tape will do..!)



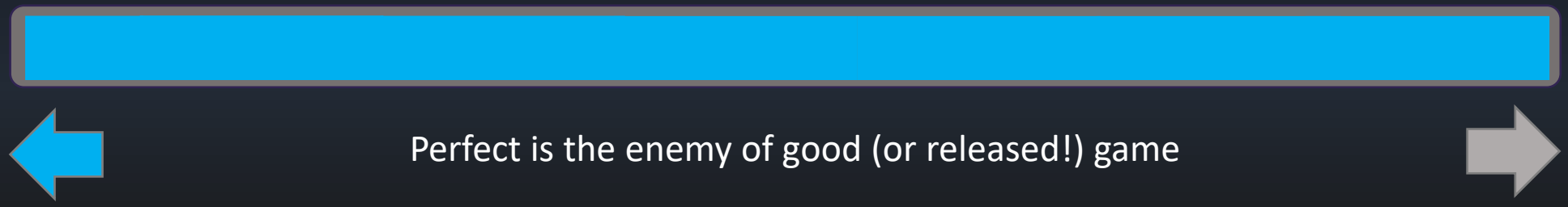
Before we start…

- Multiple techniques, we'll scratch the surface
- No generally correct approach – complex implementation != fun (or successful) game
  - When you're holding a hammer, everything looks like a nail
    - Learn to realize when you need to use a screwdriver
      - (or when duct tape will do..!)
- Evolving field, YOU could be the one to come up with a new solution



Before we start…

- Multiple techniques, we'll scratch the surface
- No generally correct approach – complex implementation != fun (or successful) game
  - When you're holding a hammer, everything looks like a nail
    - Learn to realize when you need to use a screwdriver
      - (or when duct tape will do..!)
- Evolving field, YOU could be the one to come up with a new solution
- There's always a trade-off somewhere, learn to live with it



Note: Static vs Dynamic behavior

- Techniques differ in various aspects, you could categorize along either
- E.g. how static vs. dynamic the resulting behavior is:
  - ‘Static’ = AI controlled entities always behave the same way
  - ‘Dynamic’ = AI controlled entities can surprise the player, they adjust their plans according to situation
  - As everything in life, it’s a spectrum :)

‘Scripted behavior’

‘Scripted behaviour’

- Behaviour of non-player entities is determined ahead of time
- Driven by code that describes a sequence of conditions and actions, a ‘script’
  - E.g. Tower in tower defense games – ‘If enemy is close enough, shoot’
- Complexity varies – can be used in different genres and scenarios
  - Or just specific parts of a game

‘Scripted behaviour’

- Can take the form of direct instruction set
  - Go to this location
  - Look at the player
  - Shoot them with a bazooka
  - Put on sunglasses
- Or conditions that are evaluated and trigger actions
  - If an enemy passes by close enough, shoot them with a cannon
  - Once every 5 seconds, pop up from the trench and shoot in the player’s direction
- …or a combination of either

‘Scripted behavior’

Fig. 9. Ghosts in Pac-Man use fairly simple conditions for their behaviour. Pac-Man, Namco, 1980.

‘Scripted behavior’

Fig. 10. Raids in World of Warcraft have bosses whose behaviour follows a pre-determined script. World of Warcraft, Blizzard Entertainment, 2004.

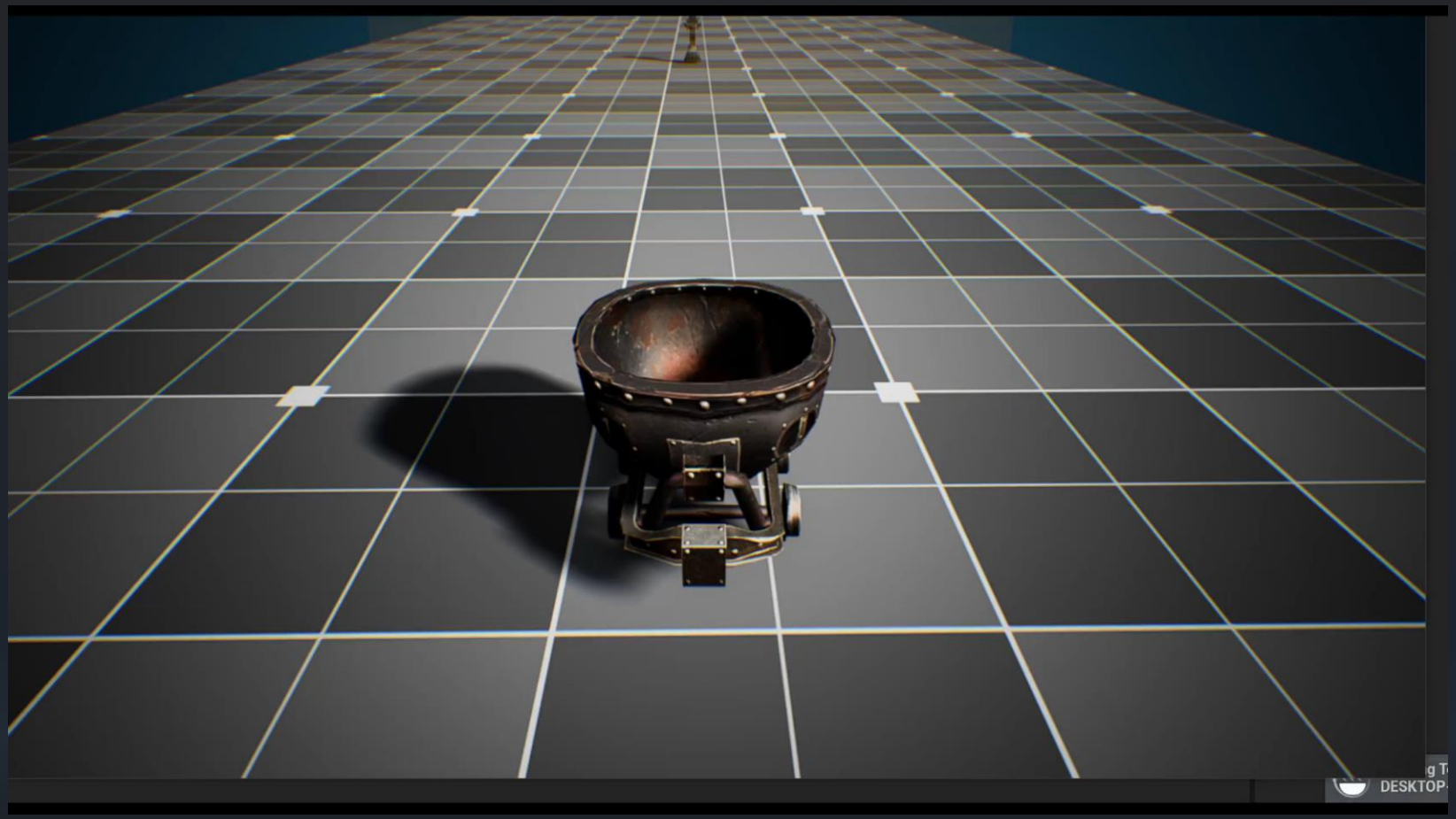
‘Scripted behavior’

Fig. 11. Enemies in a tower defense game. Kingdom Rush, Ironhide Game Studio, 2014.

‘Scripted behavior’

Fig. 12. A whole sequence can be scripted – enemy vehicles spawn from predetermined points, follow a path while the soldiers shoot the player. Call of Duty 2, Infinity Ward, 2005.

‘Scripted behavior’ – Example



‘Scripted behavior’

- Code often written in a different programming (‘scripting’) language different from the rest of the game (Lua, Python, C#, often custom solution)
  - Designer accessibility
    - Less work for programmers :)
  - Technical reason – scripts compiled separately / evaluated at runtime; faster iteration when developing game content

‘Scripted behavior’ – Advantages

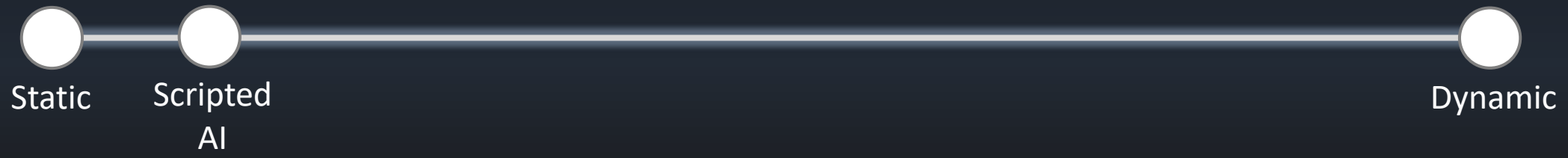
- Direct control over the game experience
  - Adjust difficulty, pacing, …
- Authored by game-designers (usually)
  - Programmers can focus on other things :)
- ‘Solvable’ by players - Can be advantageous to certain game genres
- Generally cheap, performance-wise

‘Scripted behavior’ – Disadvantages

- Amount of work!
- Game-designers can only work with what we give them :(
  - Programmers may need to expose additional functionality as scripts grow
- Predictability – limits replay-ability
- Not very responsive to player behavior
  - Behavior and the conditions to set it off need to be prepared

‘Scripted behavior’ - Overall

- Works surprisingly well for certain genres
- Usually used as a part of more complex solution
- Is this (strictly speaking) AI?
  - Debatable :)

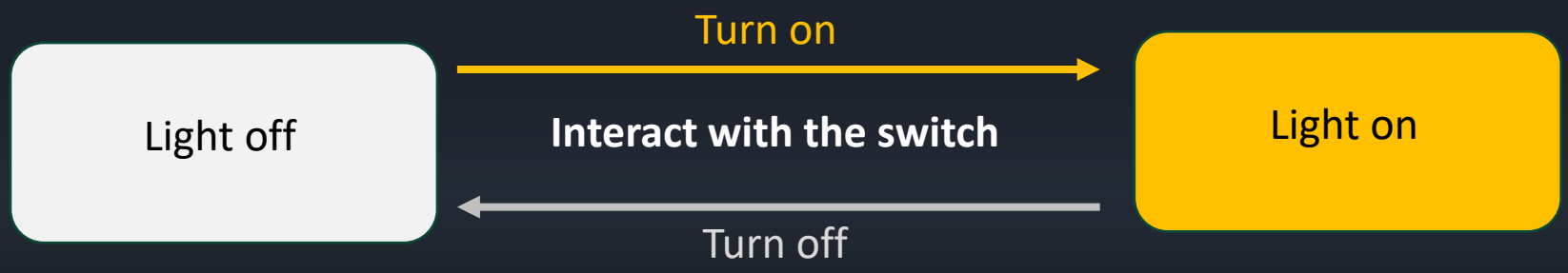


State machines

or Finite State Machines (FSM)

State machines

- A graph of states the entity can currently be in
- Defined connections/transitions between states
- Simple example – lightbulb



State machines – Example Usage

IDLE

Fig. 13. A character model 'Wraith' from an unreleased game. Paragon Asset Pack, Epic Games, ~2020.

State machines – Example Usage

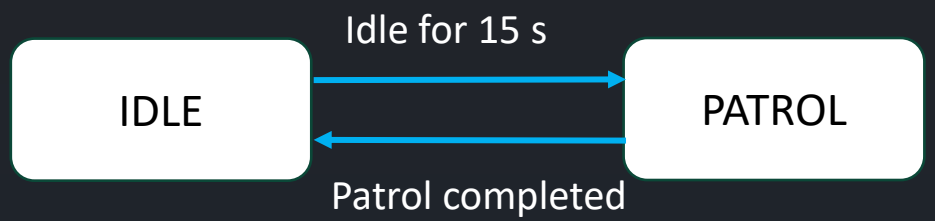


Fig. 14. Patrolling is NPC's bread and butter. Paragon Asset Pack, Epic Games, ~2020.

State machines – Example Usage

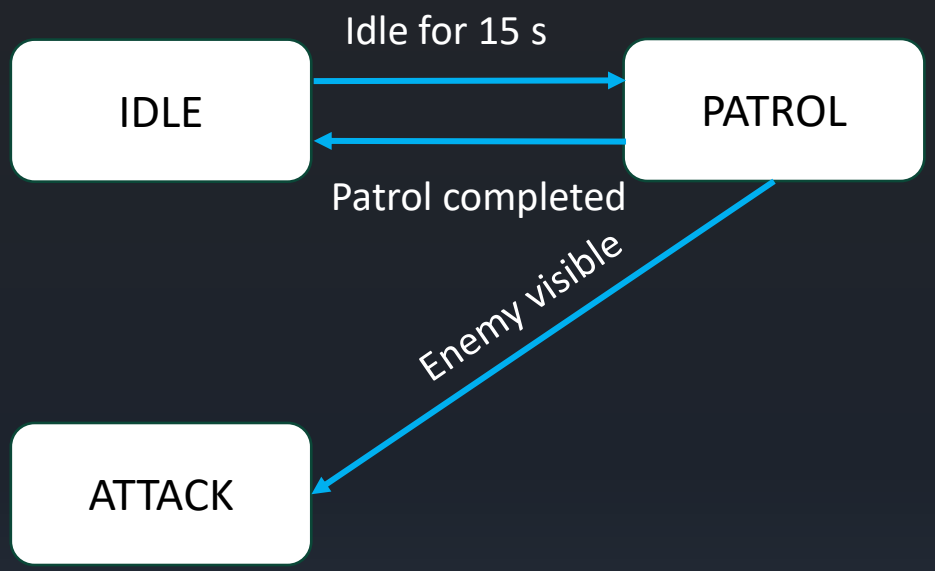


Fig. 14. Patrolling is NPC's bread and butter. Paragon Asset Pack, Epic Games, ~2020.

State machines – Example Usage

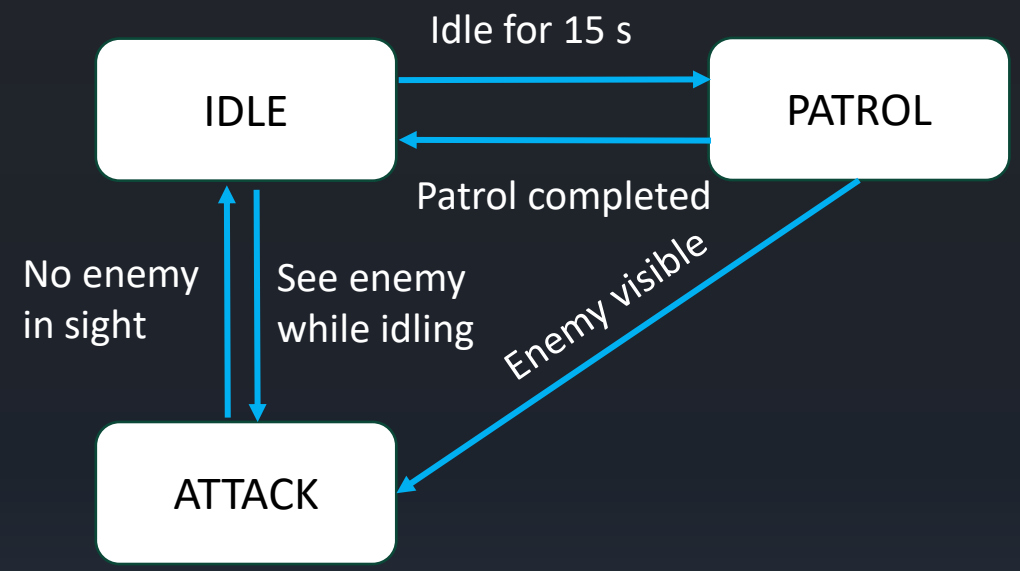
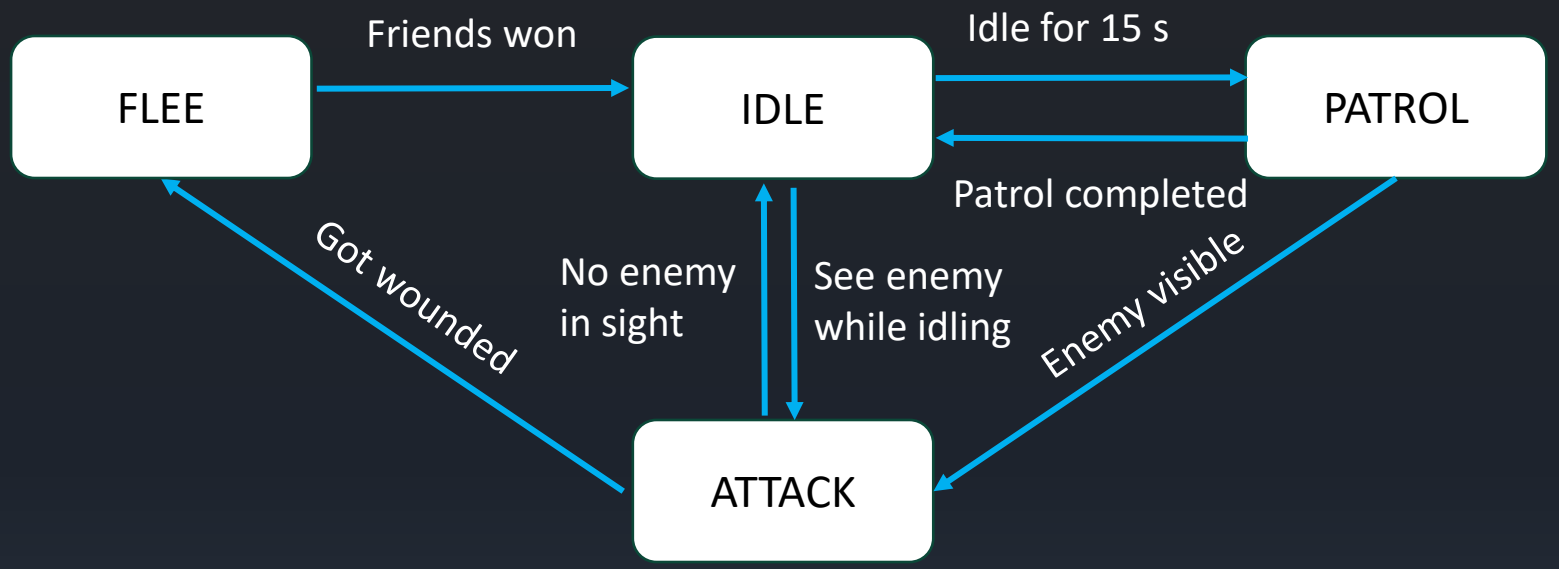


Fig. 15. This NPC woke up and chose violence. Paragon Asset Pack, Epic Games, ~2020.

State machines – Example Usage



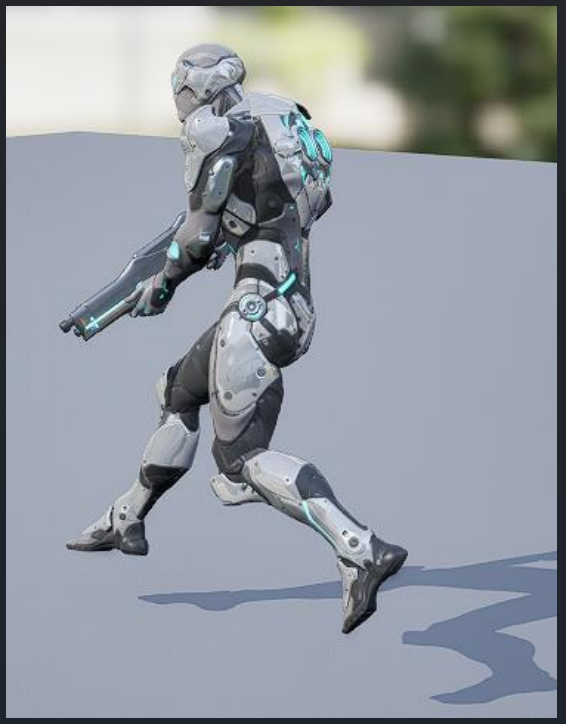
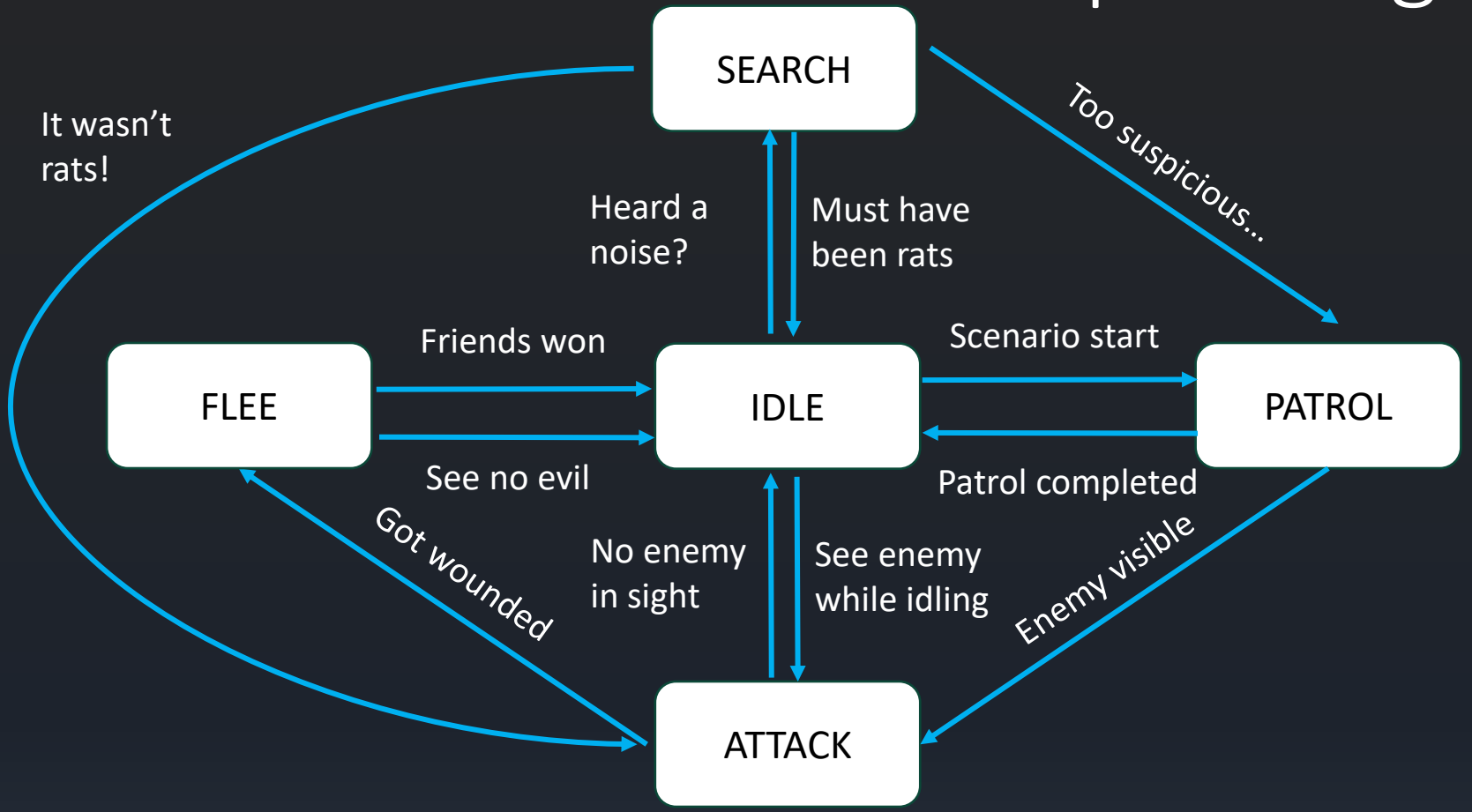


Fig. 16. Discretion is better part of valor. Paragon Asset Pack, Epic Games, ~2020.

State machines - Advantages

- More complex entity behavior
- Could still be authored by game designers :)
  - With the right support (tooling, exposing behaviors…)
- You can have graphs within graphs – see hierarchical state machines

State machines – Example Usage



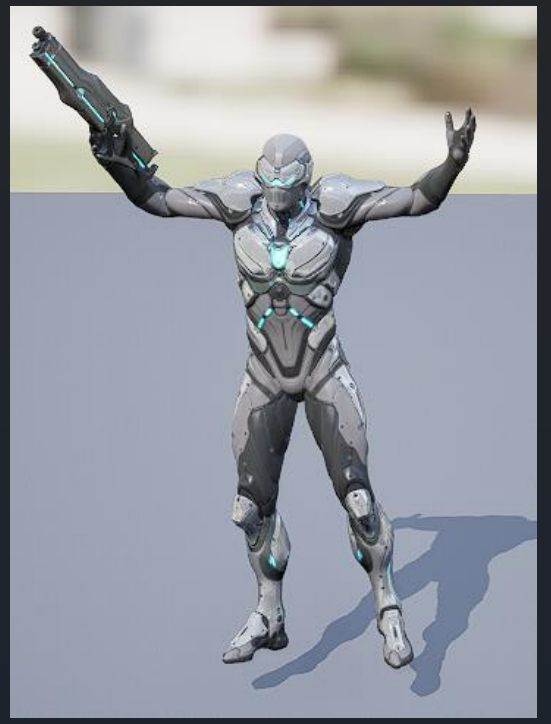


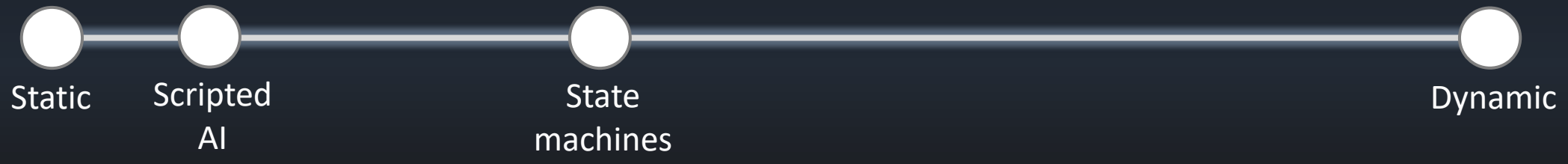
Fig. 17. It's getting pretty complex. Paragon Asset Pack, Epic Games, ~2020.

State machines - Disadvantages

- States and transitions have to be explicitly defined
  - Difficult to have things that ‘happen regardless of state’ – e.g. always flee when health is low
- Always in just one state
  - Difficult to do model things that could be feasibly done at once (run & shoot)
- Complexity-creep
  - Can be tamed with hierarchical state machines (or worsened)

State machines - Overall

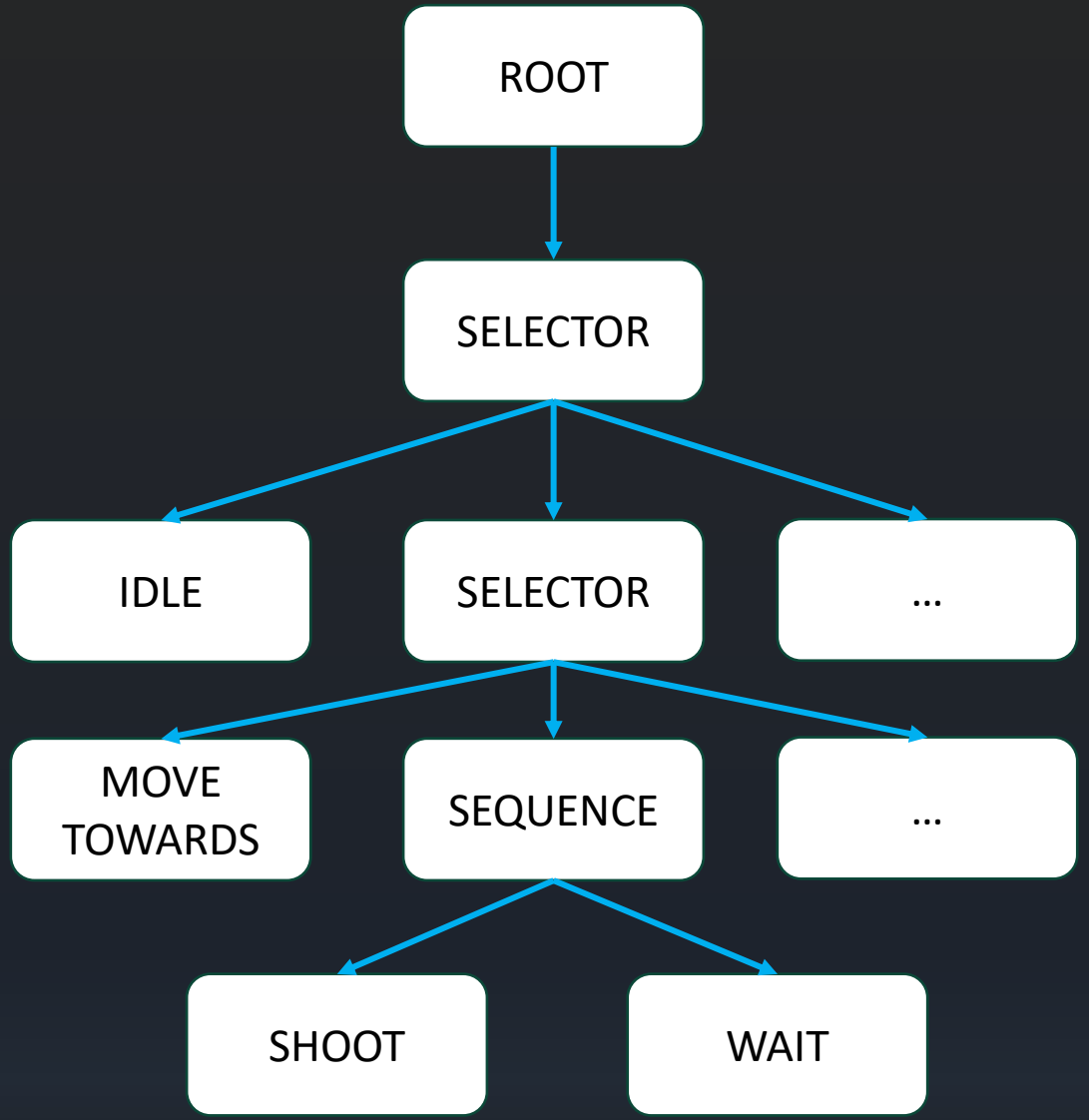
- Basic system that can work well in many games
- Doesn’t have to be complex to allow complex behaviors
  - Complexity can be manageable if hierarchies are used, states and transitions well thought out
- Can lead to fairly responsive AI entities
  - while allowing for design control



Behavior trees

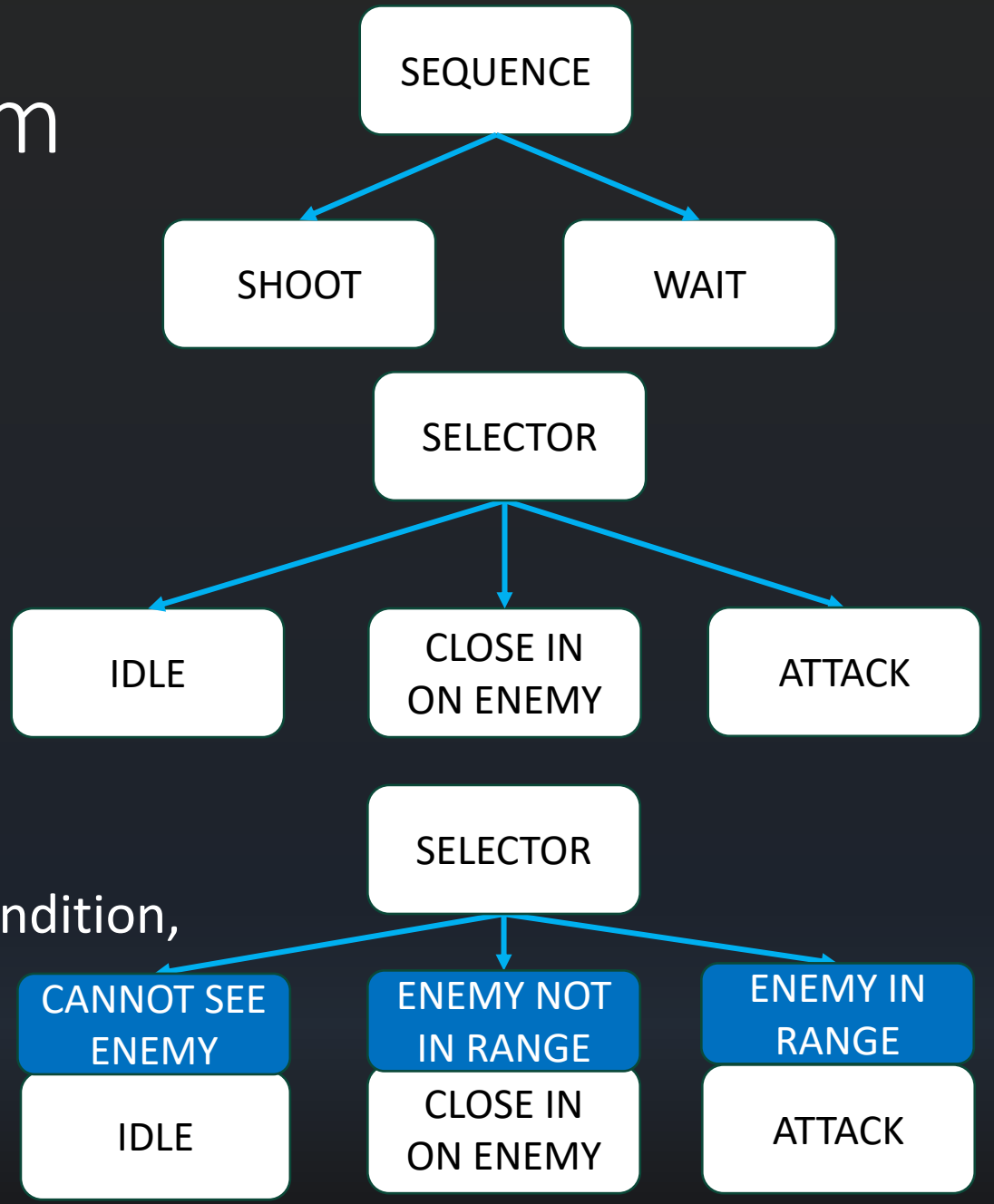
Behavior trees - Basics

- A tree of nodes that represent states or actions – each node has 0+ children nodes
- Tree is continuously evaluated from the root (traversed)
- Leaf nodes represent actions
- Transitions handled by tree structure and composite / utility / ‘meta’ nodes
- Composite nodes affect the tree traversal and the AI agent’s behavior

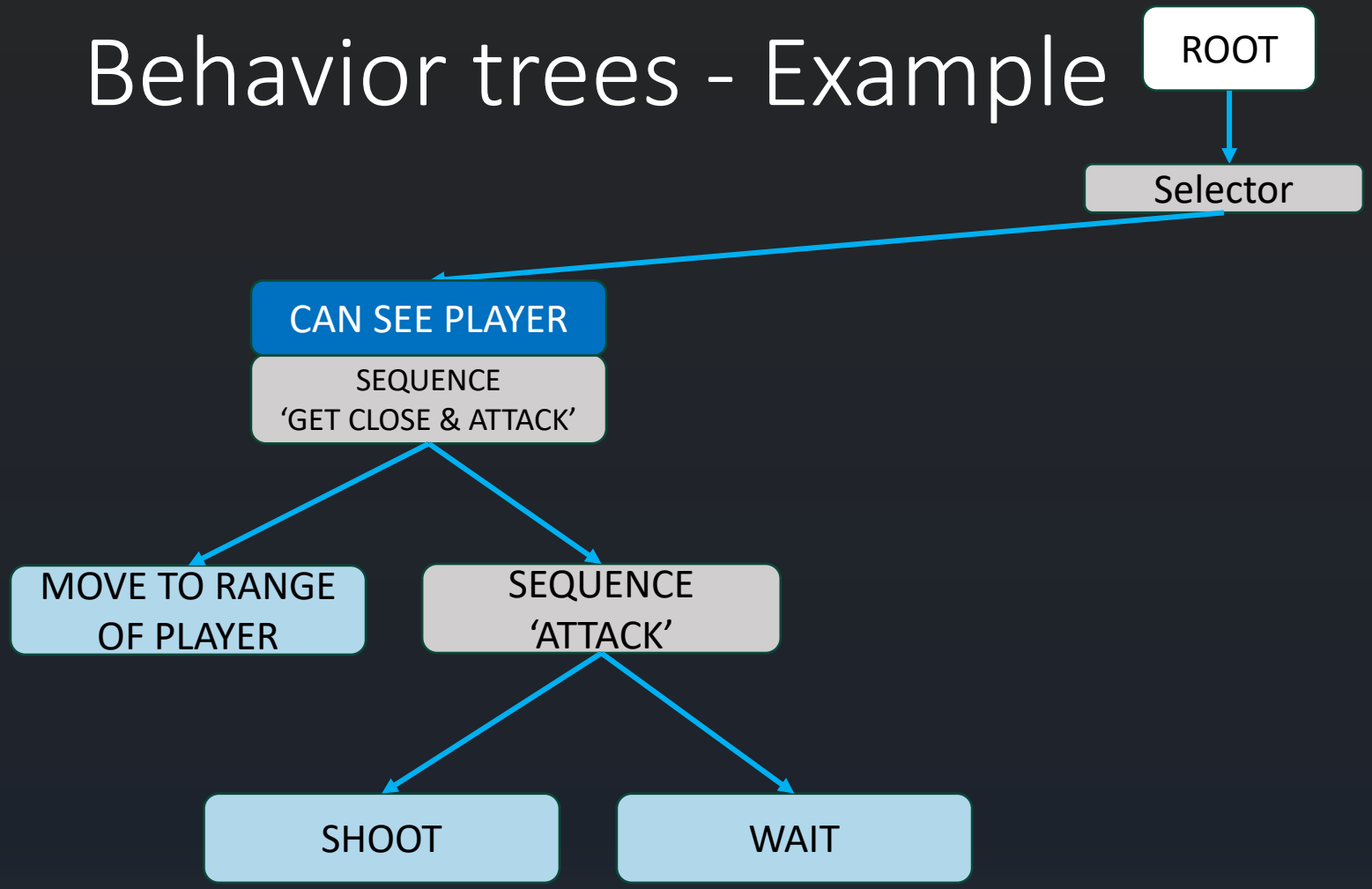


Behavior trees - Mechanism

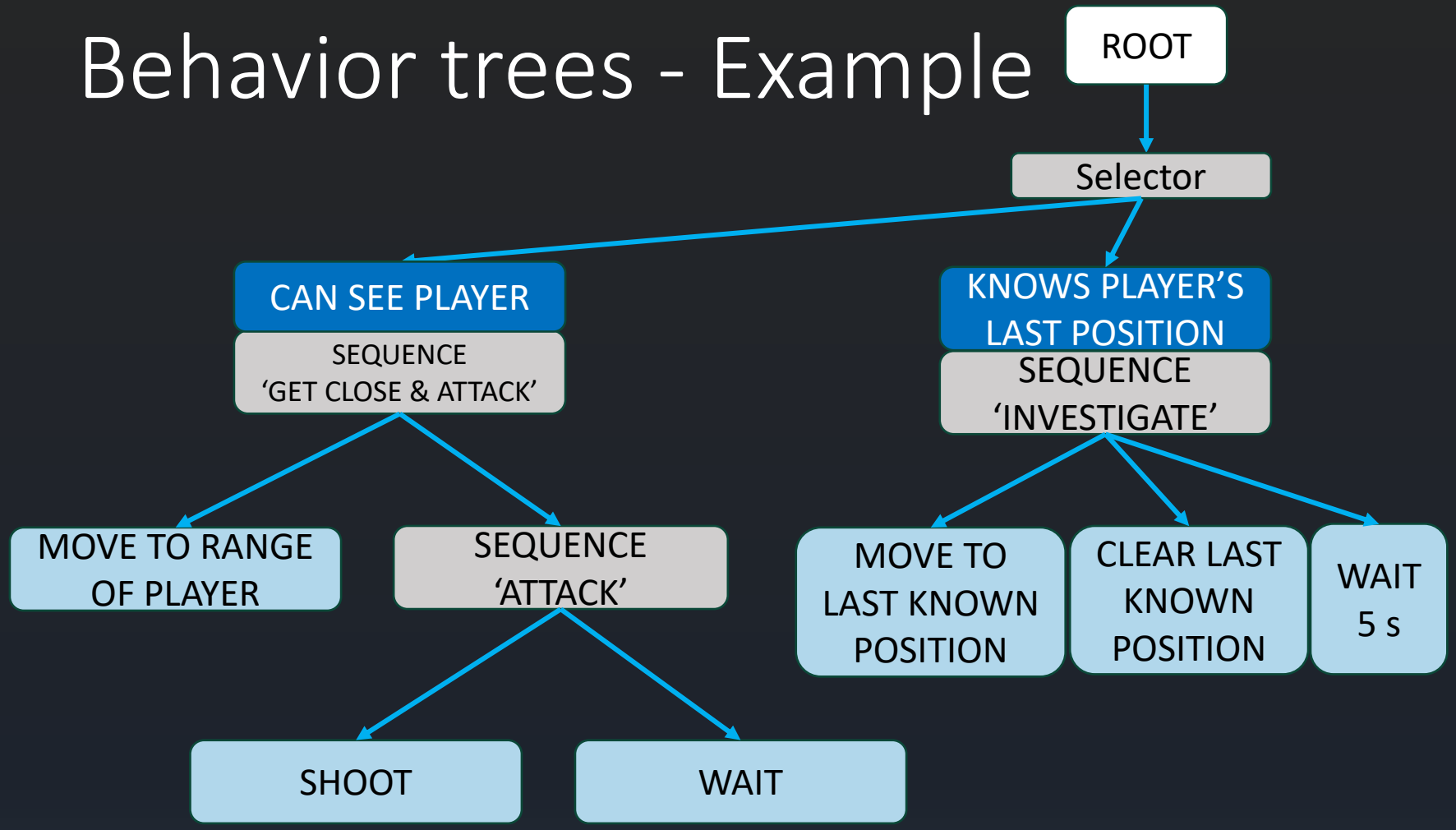
- Each node returns state
  - commonly ‘success’, ‘failure’, running’
- Sequence node
  - Evaluate children node one after the other
- Selector node
  - Choose which child node to evaluate
- Decorator nodes
  - ‘Tacked onto’ other nodes, to e.g. add a condition, store state etc.
  - Standalone nodes in some frameworks



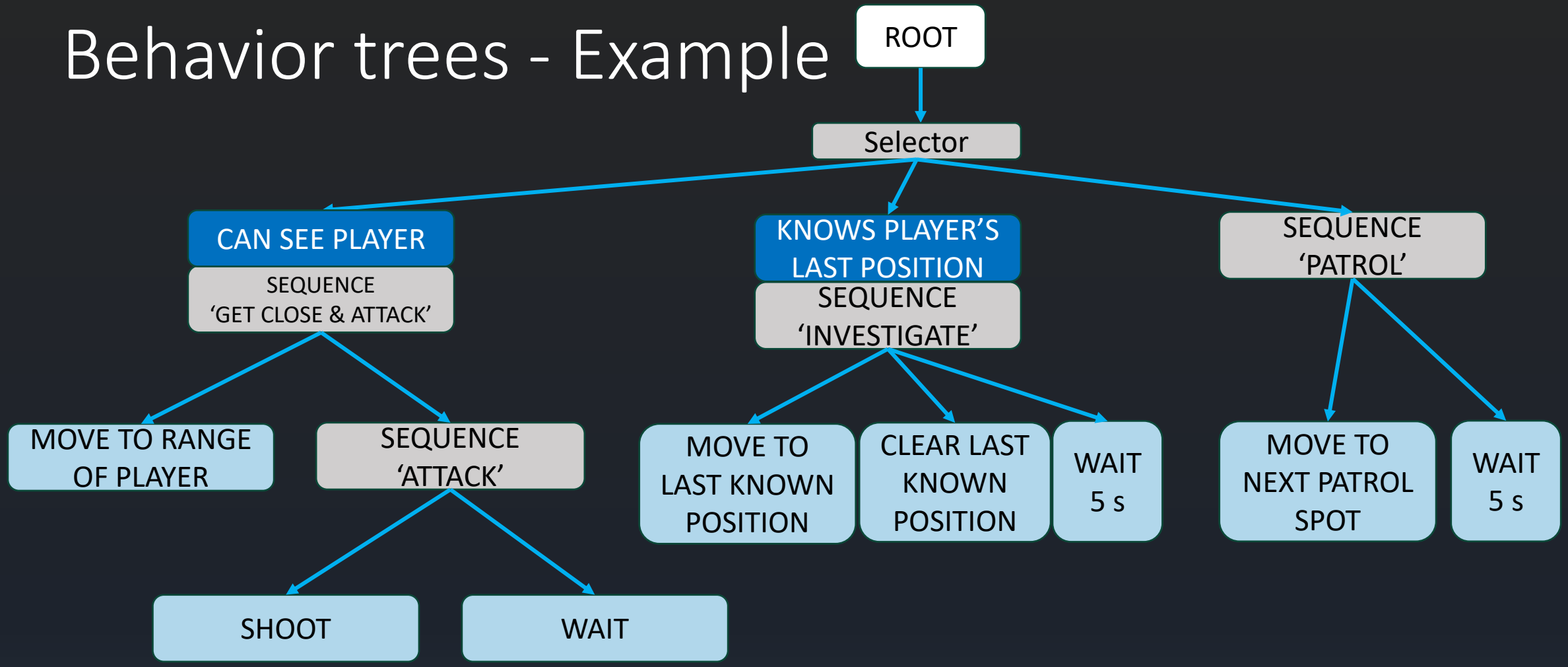
Behavior trees - Example



Behavior trees - Example



Behavior trees - Example



Behavior trees - Advantages

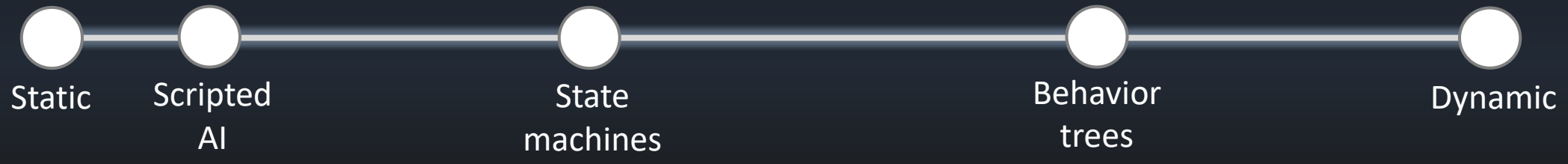
- Powerful in terms of defining behavior
- Can be defined and maintained by designers (once actions are implemented by programmers)
- Reactive AI behavior
- Game engine support
  - Most game engines will support this out of the box (e.g. Unreal, Godot, …) or have plugins (Unity)
- Fairly ‘simple’ debugging (why is the AI doing what it’s doing)
  - Usually visual representation in engine or framework

Behavior trees - Disadvantages

- Can get complex
  - It might get a bit difficult to define a tree that works well
- Can get performance ‘heavy’
  - Each agent having its own complex tree might be too much
  - Some engines/frameworks use special considerations to lessen the burden (conditional decorator checks in Unreal etc.)

Behavior trees - Overall

- Probably the most prevalent AI approach in games today
- Entities can react to changes in the game state
  - Still fairly controllable by designers (win)
    - Meaning maintenance of the AI behavior is game design's problem :)
      - (until the design team comes with a requirement for a new action or decorator)
- Works well for many use-cases
  - depending on tree structure you can get a lot of emergent behavior or fairly strict sequences of actions



Side-quest:
Utility-based approaches

Utility-based approaches

- Behavior in previous examples fairly rigidly built
  - Not quite how humans work
    - Decisions we make depend on a lot of factors, often very subjective
- If we could model this, we'd be closer to a more 'human' behaving AI entity
  - Believable choices and behavior
  - Potentially emergent behavior => interesting solutions to problems
    - Or interesting new problems... (e.g. drunk cats in Dwarf Fortress)

Utility-based approaches

- Each decision the AI can make is given a utility score, affecting the ‘choices’ the AI makes

- The score is re-evaluated as the game state changes

SLEEP	0.3
EAT	0.2

ENERGY 0.25

SLEEP	0.8
EAT	0.1

Fig. 18. Sometimes sleep is a priority. Sims 4, Maxis, 2014.

Utility-based approaches

- Each decision the AI can make is given a utility score, affecting the ‘choices’ the AI makes

- The score is re-evaluated as the game state changes
  - Effect doesn’t have to be linear
  - Multiple factors can be combined
    - E.g. entity is hungry but has no food to eat – eat action utility score will be zero

SLEEP	0.3
EAT	0.2

ENERGY 0.25		SATIETY 0.20	
SLEEP	0.8	SLEEP	0.2
EAT	0.1	EAT	0.7

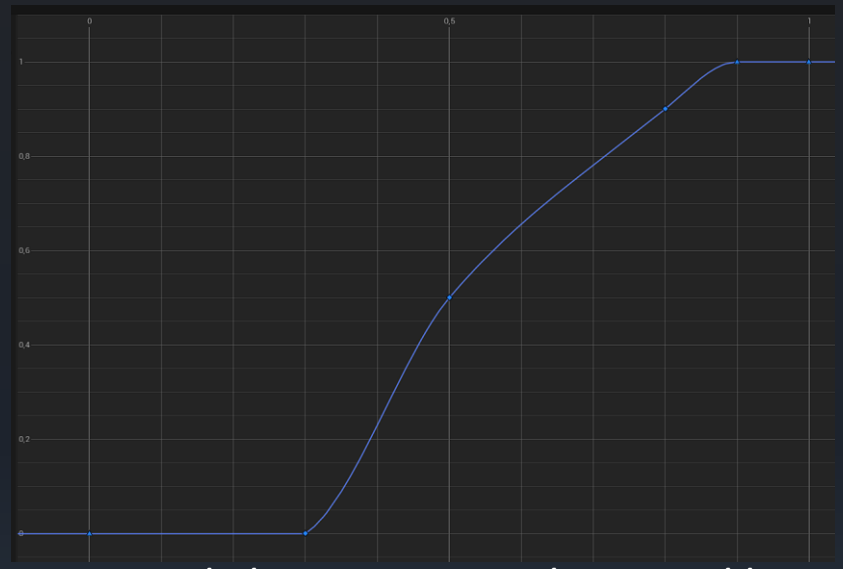


Fig. 19. Example hunger curve where until hunger hits a certain threshold, it has no effect at all – AI won’t consider eating when it’s barely hungry (x = normalized hunger value, y = utility score).

Utility-based approaches - Overall

- Usually used as part of a more complex solution for AI

Side-quest completed

Goal Oriented Action Planning (GOAP)

Goal Oriented Action Planning - Basics

- An AI solution where we leave the choice of what the entity should do (and how) in a given moment to the AI system

- Decision making is done during run-time
  - AI ‘planner’ builds a plan of actions based on the state of the in-game world, then acts on it

- ‘Utility’ approach is used on multiple levels (choosing actions, choosing plan)

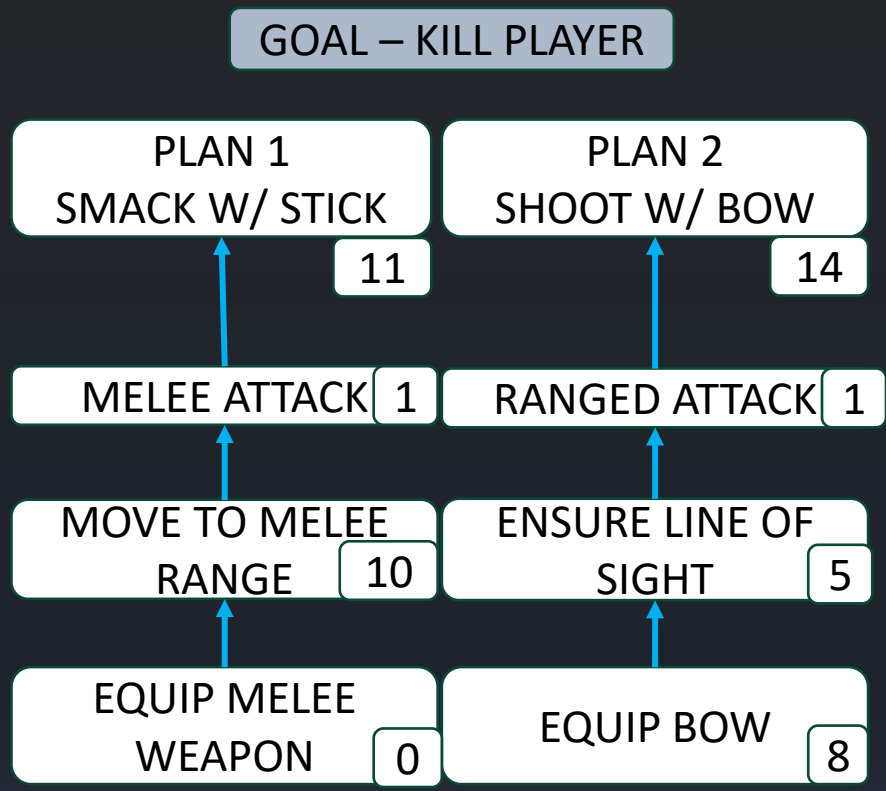
Goal Oriented Action Planning - Mechanism

- AI chooses a Goal, then builds a Plan from Actions

- Goal
  - what the AI wants to ultimately do
- Plan
  - a sequence of actions that should allow the AI to reach its Goal
- Action
  - an in-game action the AI entity can do; each has it's own cost

- Actions that take us closer to the Goal are selected (utility) and a plan is created
  - Plan built from the end
  - Plan score can be simply a sum of action costs

- Rinse and repeat – then choose the best plan (utility score of actions)



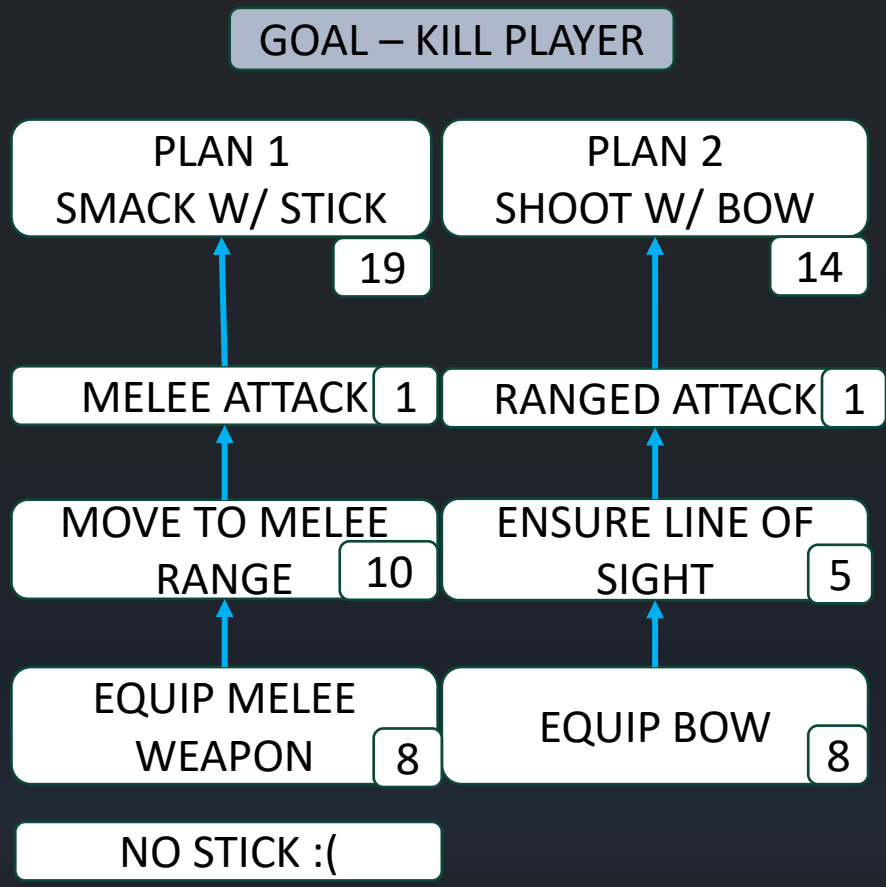
Goal Oriented Action Planning - Mechanism

- AI chooses a Goal, then builds a Plan from Actions

- Goal
  - what the AI wants to ultimately do
- Plan
  - a sequence of actions that should allow the AI to reach its Goal
- Action
  - an in-game action the AI entity can do; each has it's own cost

- Actions that take us closer to the Goal are selected (utility) and a plan is created
  - Plan built from the end
  - Plan score can be simply a sum of action costs

- Rinse and repeat – then choose the best plan (utility score of actions)
- Action costs can be conditional



Goal Oriented Action Planning – Game Examples

Fig. 20. The game where GOAP made its debut. F.E.A.R., 2005, Monolith Productions.

Goal Oriented Action Planning – Game Examples

Fig. 21. Middle-earth: Shadow of Mordor, 2014, Monolith Productions.

Goal Oriented Action Planning – Game Examples

Fig. 22. Tomb Raider, 2013, Monolith Productions.

Goal Oriented Action Planning - Advantages

- Given action building blocks, AI agent can achieve a lot of goals
- Entity should be naturally trying to do things that bring it closer to its Goal => it will do things the player might do in their shoes!
  - E.g. An NPC in a turn-based RPG game wants to attack our player character
    - But our character is just outside of their range of movement
    - What if they have a potion of speed in their pockets…
    - They can use that, move closer and smack the player (with a stick)!
- Usable for different genre of games (not only combat-oriented)

Goal Oriented Action Planning - Disadvantages

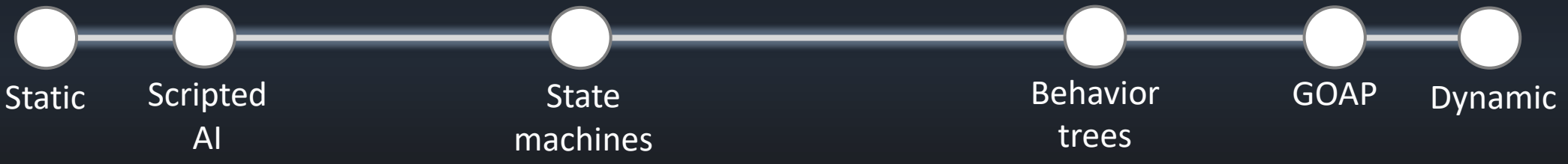
- Performance heavy
  - AI will prefer the simplest plan (lowest cost / highest utility score) but can still build multiple
  - And doesn’t scale too well (more actions => longer building; more AI entities => costly)
- May require additional work on the game content (tagging etc.)
  - so that AI can interact with the world (e.g. tagging certain entities as ‘doors’, adding cover points…)
- Design complexity
  - Designers can only affect behavior indirectly, via utility costs of actions and other means; they might have issues ‘reading the state’
- Debugging ‘fairly complex’
  - Depends on your tooling and/or skillset but might require a programmer

Goal Oriented Action Planning - Overall

- Leads to systemic behavior => dynamic gameplay => more replay value (than statically scripted game sections)

- Can run into problems at scale
  - Building plans is a lot of calculations, so too many entities can be problematic

- Can lead to very interesting gameplay choices



Wrap up!

- None of the solutions represented is a ‘silver bullet’
  - You could use multiple or neither in a successful game
- Data-driven is the way to go
  - Leave the ‘grunt’ work to game/content designers! :)

Further topics for those interested

- Game AI systems/architectures
  - Hierarchical Task Network Planning (HTN)
  - Hierarchical State Machines
- Other interesting topics
  - Pathfinding
  - Nav-mesh building (and simplification)
  - Entity detection, collision detection

References

- Fenlon, W. (2016, March 31). How cats get drunk in Dwarf Fortress, and why its creators haven't figured out time travel (yet). *Pcgamer*. https://www.pcgamer.com/how-cats-get-drunk-in-dwarf-fortress-and-why-its-creators-havent-figured-out-time-travel-yet/
- GDC. (2017, October 9). *Goal-Oriented Action Planning: Ten years of AI programming* [Video]. YouTube. https://www.youtube.com/watch?v=gm7K68663rA

Thank you!